In this lecture, we will use the following statements, which you proved as an exercise.

Lemma 1. *Let f be a flow in an s-t-network $\mathcal{N} = (V, E, c, s, t)$, and let g be a flow be the residual network $\mathcal{N}_f$ of $\mathcal{N}$ w.r.t. f. Then, the following statements are true.*

(a) $f + g$ is a flow in $\mathcal{N}$.

(b) If g is a maximum flow in $\mathcal{N}_f$, then $f + g$ is a maximum flow in $\mathcal{N}$.

Lemma 2. *Let $\mathcal{N} = (V, E, c, s, t)$ be an s-t-network with m edges. Then, there exists a maximum flow in $\mathcal{N}$ that is the sum of at most m flows $f_1, \dots, f_k$, each of which takes positive values only on a single s-t-path in $\mathcal{N}$. Moreover, if all capacities in $\mathcal{N}$ are integers, then $f_1, \dots, f_k$ can be chosen as integral flows.*

1 Dinitz' Algorithm

In the first part of this lecture, we introduce and analyze Dinitz' algorithm for computing a maximum flow in an s-t-network. The algorithm of Edmonds and Karp uses $O(m)$ time to find an augmenting path in the residual graph via BFS. However, we can intuitively find multiple augmenting paths in the residual graph in such a single BFS traversal. This is the main idea of Dinitz' algorithm, and roughly reduces the time per path to $O(n)$. To do so, we will repeatedly find *blocking flows* in the residual graph instead of augmenting paths.

Definition 1 (Blocking Flow)**.** *A* blocking flow *in an s-t-network $\mathcal{N} = (V, E, c, s, t)$ is a feasible flow that saturates at least one edge on every s-t-path in $\mathcal{N}$.*

An example of a blocking flow is shown in Figure 1.

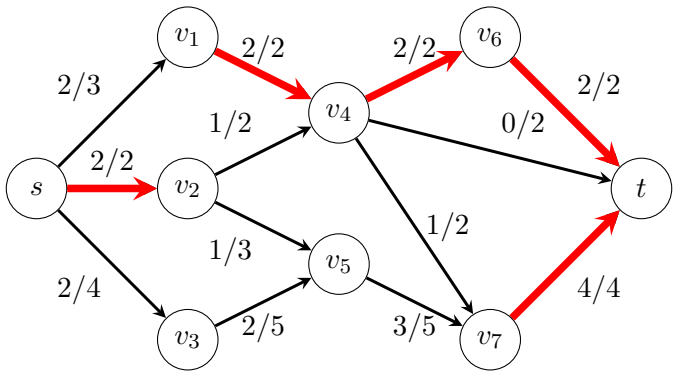

Figure 1: An example of a blocking flow in an s-t-network. The numbers on the edges represent flow/capacity. Saturated (blocked) edges are highlighted in red. Note that the flow is not a maximum flow.

It is easy to see that every acyclic s-t-network has a blocking flow. Unfortunately, the residual graph of a flow network is not necessarily acyclic. However, we can still find a blocking flow in the residual graph by using a *layered network*. To this end, we define for every vertex $v \in V$ and a flow f the *height* $\delta_f(v)$ as the distance (number of edges) of a shortest path from v to t in $\mathcal{N}_f$ (we assume that $\delta_f(v) = \infty$ if there is no such path).

Definition 2 (Layered Network)**.** *Let $\mathcal{N} = (V, E, c, s, t)$ be an s-t-network. The* layered network *of $\mathcal{N}$ w.r.t. f is the s-t-network $\mathcal{N}' = (V, E', c', s, t)$, where $E' = \{(u, v) \in E_f \mid \delta_f(v) < \delta_f(u)\}$ and c' is the restriction of c_f to E'. For an $i \in \mathbb{N}_0$, the* layer *i of $\mathcal{N}'$ is the set of vertices $V_i = \{v \in V \mid \delta_f(v) = i\}$.*

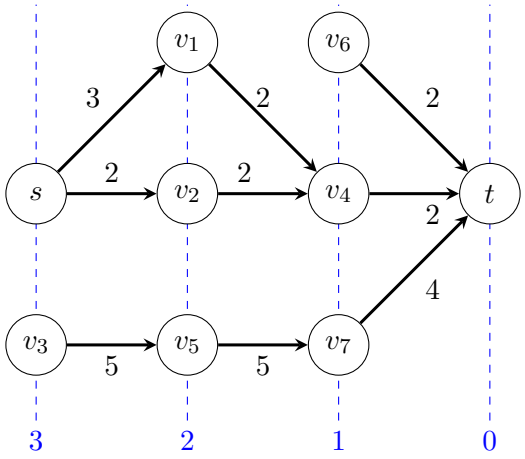

Figure 2: Layered network corresponding to the network in Figure 1 with respect to $f = 0$. The dashed blue lines indicate the layers of the network.

An example of a layered network is shown in Figure 2. Note that every layered network is acyclic, since every edge $(u, v) \in E'$ satisfies $\delta_f(v) < \delta_f(u)$. Thus, there exists a blocking flow in the layered network $\mathcal{N}'$ of $\mathcal{N}$ w.r.t. f.

Lemma 3. *A blocking flow in an acyclic s-t-network $\mathcal{N} = (V, E, c, s, t)$ with n vertices and m edges be computed in $O(nm)$ time.*

Proof. Consider the following algorithm:

- Repeatedly start a DFS from s until reaching t.
- Whenever the DFS retreats over an edge e, remove e.
- Whenever reaching t, carry out an augmentation along the found s-t-path. Remove all edges that become saturated by the augmentation.
- Stop when t is unreachable from s.

When the algorithm removes an edge (u, v), either (u, v) is saturated, every path from v to t in the original network (without removed edges) contains a saturated edge. The latter can be proven by induction on the number of removed edges. Thus, the algorithm computes indeed a blocking flow.

It remains to analyze the running time. The algorithm spends $O(m)$ time retreating over and removing edges. After at most $n - 1$ steps, the search much find t, and thus, saturate and remove at least one edge. Therefore, the total time is in $O(nm)$. □

We are now ready to state Dinitz' algorithm in Algorithm 1. As explained above, we will repeatedly compute blocking flows in the residual graph. The algorithm terminates when no more blocking flow can be found.

The algorithm terminates when no more blocking flow can be found, which is the case when there is no path from s to t in the residual graph. Moreover, by Lemma 1, the algorithm computes a feasible flow, and thus, is correct.

We will now analyze the running time of Dinitz' algorithm.

Lemma 4. *Every iteration of Dinitz' algorithm (line 2) strictly increases $\delta_f(s)$.*

Algorithm 1: Dinitz' Algorithm

Input: An s-t-network $\mathcal{N} = (V, E, c, s, t)$
Output: A maximum flow f^* in $\mathcal{N}$

```
f ← 0;
while f is not maximum do
    h := a blocking flow in the layered network of N w.r.t. f;
    f ← f + h;
return f;
```

Proof. Consider an iteration of Dinitz' algorithm. Let f and f' be the flows in $\mathcal{N}$ before and after the iteration, respectively. Let $\mathcal{L}$ be the layered network of $\mathcal{N}$ w.r.t. f.

Let (v, w) be an edge in $\mathcal{N}_{f'}$. We consider two cases. If (v, w) is also an edge in $\mathcal{N}_f$, then we have $\delta_f(v) \leq \delta_f(w) + 1$ by the triangle inequality of shortest paths. Otherwise, the algorithm sends flow algorithm (w, v) in that iteration, and thus, (w, v) is an edge in $\mathcal{L}$. By the definition of a layered network, we have $\delta_f(v) < \delta_f(w)$. In either case, we have $\delta_f(v) \leq \delta_f(w) + 1$.

If $\delta_{f'}(s) = \infty$, then $\delta_{f'}(s) > \delta_f(s)$ and we are done. Otherwise, let $k := \delta_{f'}(s)$ and $P = (v_k, v_{k-1}, \ldots, v_0)$ be a shortest path from s to t in $\mathcal{N}_{f'}$, where $v_k = s$ and $v_0 = t$. By the previous inequality, we deduce $\delta_f(v_k) \leq \delta_f(v_{k-1}) + 1 \leq \ldots \leq \delta_f(v_0) + k$. Since $\delta_f(v_0) = 0$, we have $\delta_f(s) \leq k = \delta_{f'}(s)$.

To conclude the lemma, assume that $\delta_f(s) = k$. Thus, we must have $\delta_f(v_i) = \delta_f(v_{i-1}) + 1$ for $i = 1, \ldots, k$. This means that every edge on P is part of $\mathcal{L}$. But since the fixed iteration saturates at least one edge on P with the blocking flow h in $\mathcal{L}$, we have a contradiction to P being a path in $\mathcal{N}_{f'}$. Thus, we conclude that $\delta_f(s) < \delta_{f'}(s)$. □

Since the height of s can be at least 1 and at most $n-1$ until the last iteration (where it changes to ∞), we conclude that the number of iterations of Dinitz' algorithm is at most $n-1$. Together with Lemma 3, we conclude the following theorem.

Theorem 1. *Dinitz' algorithm computes a maximum flow in an s-t-network with n vertices and m edges in $O(n^2m)$ time.*

2 Simple Networks and Faster Bipartite Matching

In the first lecture of this course, we have seen that we can compute a maximum matching in a bipartite graph in $O(mn)$ time, where n is the number of vertices and m is the number of edges.

We will now show how to improve this running time to $O(m\sqrt{n})$ by using ideas from Dinitz' algorithm. If we simply use the reduction of maximum bipartite matching to maximum flows, our analysis of Dinitz' algorithm would only give a running time of $O(n^2m)$. However, the network that arises from the reduction from a bipartite matching instance has a special structure, which we can exploit to give an improved bound on the running time. In particular, such a network is *simple*.

Definition 3 (Simple s-t-network)**.** *An s-t-network $\mathcal{N} = (V, E, c, s, t)$ is* simple *if all capacities are integers, and for every $v \in V \setminus \{s, t\}$, at least one of the following conditions holds:*

(i) the capacities of the edges entering v sum up to at most 1, or

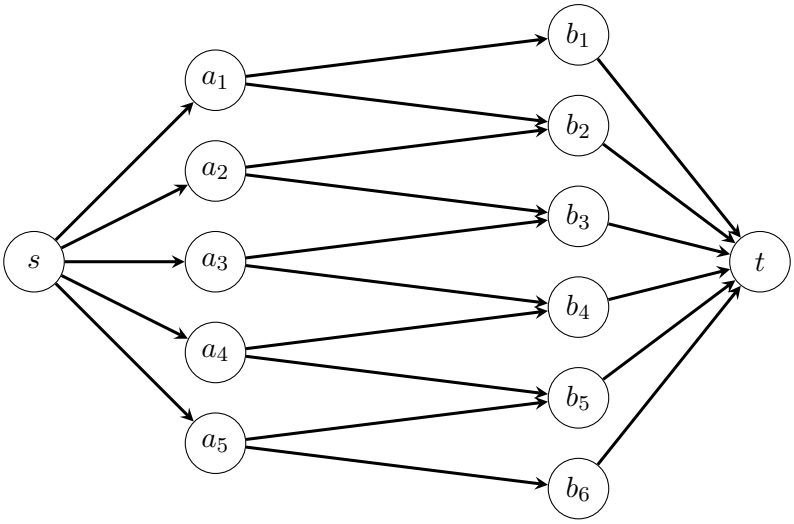

Figure 3: Reduction of a bipartite matching instance to a simple s-t-network. All capacities are equal to 1.

(ii) the capacities of the edges leaving v sum up to at most 1.

In other words, a simple network has at most one unit of flow entering or leaving every vertex (except for s and t). Given a bipartite graph $G = (A, B, E)$, we can use the reduction from the first lecture to construct a simple network $\mathcal{N} = (V, E, c, s, t)$, where $V = \{s, t\} \cup A \cup B$. Then, every vertex $a \in A$ satisfies the first condition, and every vertex $b \in B$ satisfies the second condition in Definition 3. An example in given in Figure 3.

Thus, we now focus on simple networks. We first show the useful property that the residual network of a simple network is also simple.

Lemma 5. *The residual network $\mathcal{N}_f$ with respect to an integral flow f of a simple s-t-network $\mathcal{N}$ is also simple.*

Proof. Let $\mathcal{N} = (V, E, c, s, t)$ be a simple s-t-network, and let $v \in V \setminus \{s, t\}$, and let f be an integral flow in $\mathcal{N}$.

Assume that v satisfies the first condition in Definition 3. If f sends no flow via v, then clearly the first condition is satisfied for v in $\mathcal{N}_f$. Otherwise, there exists a single edge e with a capacity of 1 that is entering v in $\mathcal{N}$, and since f is integral, e is saturated. By flow conservation, there exists a single distinct edge e' leaving v in $\mathcal{N}$ with $f(e') = 1$. Since e' is the only edge leaving v with positive flow and e is saturated, the only edge entering v in $\mathcal{N}_f$ is $\overleftarrow{e'}$ with a capacity of $f(e') = 1$, satisfying the first condition in Definition 3 for v in $\mathcal{N}_f$.

The case where v satisfies the second condition in Definition 3 in $\mathcal{N}$ is similar. □

We will now show that Dinitz' algorithm terminates after at most $O(\sqrt{n})$ iterations when applied to a simple s-t-network. The main idea is that by Lemma 4 after sufficiently many iterations, every s-t-path in the layered network must be quite long. Using the property that the residual network is simple, we can show that a blocking flow cannot saturate too many edges, and thus, we are already close to the maximum flow.

Lemma 6. *Dinitz' algorithm terminates after at most $O(\sqrt{n})$ iterations when applied to a simple s-t-network.*

Proof. Let $\mathcal{N} = (V, E, c, s, t)$ be a simple s-t-network, and let f be the flow after $\lceil \sqrt{n} \rceil$ iterations of Dinitz' algorithm (or the final flow if fewer iterations are performed). Lemma 4 thus implies $\delta_f(s) \geq \sqrt{n} + 1$. By Lemma 2, $\mathcal{N}_f$ has a maximum integral flow g that is the sum of several non-zero flows $g_1, \ldots, g_k$, each of which takes positive values only on a single s-t-path in $\mathcal{N}_f$. Since $\mathcal{N}_f$ is simple by Lemma 5, we conclude that these paths must be internally vertex-disjoint (that is, they are vertex-disjoint except for s and t). Moreover, as $\delta_f(s) \geq \sqrt{n} + 1$, every path has at least $\sqrt{n}$ internal vertices that are not shared with any other path. Since there are only n vertices in total, there can be at most $n/\sqrt{n} = \sqrt{n}$ such paths. Thus, $val(g) \leq \sqrt{n}$. By Lemma 1, the value of a maximum flow in $\mathcal{N}$ is $val(f) + val(g) \leq val(f) + \sqrt{n}$. Since the value of f increases in every iteration by at least 1, we conclude that the number of iterations following the first $\lceil \sqrt{n} \rceil$ iterations is at most $\sqrt{n}$. □

Moreover, a single iteration runs faster in simple networks.

Lemma 7. *A blocking flow in the layered network $\mathcal{L}$ of a simple s-t-network $\mathcal{N}$ w.r.t. an integral flow f in $\mathcal{N}$ can be computed in $O(n + m)$ time.*

Proof. Consider the same algorithm as in Lemma 3:

- Repeatedly start a DFS from s until reaching t.
- Whenever the DFS retreats over an edge e, remove e.
- Whenever reaching t, carry out an augmentation along the found s-t-path. Remove all edges that become saturated by the augmentation.
- Stop when t is unreachable from s.

First, note that $\mathcal{L}$ is also simple. Thus, on every s-t-path in $\mathcal{L}$ found by the algorithm, of every two successive edges, at least one has capacity 1. Thus, if such a path consists of k edges, the algorithm saturates and removes at least $(k-1)/2$ edges of that path. This means that the number of edges of all augmenting paths used by the algorithm throughout its execution is $O(m)$. Thus, the overall running time is $O(n + m)$. □

Lemmas 6 and 7 imply the following main theorem.

Theorem 2. *Dinitz' algorithm computes a maximum flow in a simple s-t-network with n vertices and m edges in $O(m \cdot \sqrt{n})$ time.*

From our previous discussion, we can also conclude the following corollary.

Corollary 1. *A maximum matching in a bipartite graph with n vertices and m edges can be computed in $O(m \cdot \sqrt{n})$ time.*